Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike

Jongejan, Bart; Dalianis, Hercules

Published in:
Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP

Publication date:
2009

Document version
Publisher's PDF, also known as Version of record

Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike

Bart Jongejan
CST-University of Copenhagen
Njalsgade 140-142 2300 København S
Denmark
bartj@hum.ku.dk

Hercules Dalianis† ‡
†DSV, KTH - Stockholm University
Forum 100, 164 40 Kista, Sweden
‡Euroling AB, SiteSeeker
Igeldammsgatan 22c
112 49 Stockholm, Sweden
hercules@dsv.su.se

Abstract

We propose a method to automatically train lemmatization rules that handle prefix, infix and suffix changes to generate the lemma from the full form of a word. We explain how the lemmatization rules are created and how the lemmatizer works. We trained this lemmatizer on Danish, Dutch, English, German, Greek, Icelandic, Norwegian, Polish, Slovene and Swedish full form-lemma pairs respectively. We obtained significant improvements of 24 percent for Polish, 2.3 percent for Dutch, 1.5 percent for English, 1.2 percent for German and 1.0 percent for Swedish compared to plain suffix lemmatization using a suffix-only lemmatizer. Icelandic deteriorated with 1.9 percent. We also made an observation regarding the number of produced lemmatization rules as a function of the number of training pairs.

1 Introduction

Lemmatizers and stemmers are valuable human language technology tools to improve precision and recall in an information retrieval setting. For example, stemming and lemmatization make it possible to match a query in one morphological form with a word in a document in another morphological form. Lemmatizers can also be used in lexicography to find new words in text material, including the words' frequency of use. Other applications are creation of index lists for book indexes as well as key word lists

Lemmatization is the process of reducing a word to its base form, normally the dictionary look-up form (lemma) of the word. A trivial way to do this is by dictionary look-up. More advanced systems use hand crafted or automatically generated transformation rules that look at the surface form of the word and attempt to produce the correct base form by replacing all or parts of the word.

Stemming conflates a word to its stem. A stem does not have to be the lemma of the word, but can be any trait that is shared between a group of words, so that even the group membership itself can be regarded as the group's stem.

The most famous stemmer is the Porter Stemmer for English (Porter 1980). This stemmer removes around 60 different suffixes, using rewriting rules in two steps.

The paper is structured as follows: section 2 discusses related work, section 3 explains what the new algorithm is supposed to do, section 4 describes some details of the new algorithm, section 5 evaluates the results, conclusions are drawn in section 6, and finally in section 7 we mention plans for further tests and improvements.

2 Related work

There have been some attempts in creating stemmers or lemmatizers automatically. Ekmekçioglu *et al*. (1996) have used *N*-gram matching for Turkish that gave slightly better results than regular rule based stemming. Theron and Cloete (1997) learned two-level rules for English, Xhosa and Afrikaans, but only single character insertions, replacements and additions were allowed. Oard *et al*. (2001) used a language independent stemming technique in a dictionary based cross language information retrieval experiment for German, French and Italian where English was the search language. A four stage backoff strategy for improving recall was intro-

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 145–153,
Suntec, Singapore, 2-7 August 2009.

duced. The system worked fine for French but not so well for Italian and German. Majumder *et al*. (2007) describe a statistical stemmer, YASS (Yet Another Suffix Stripper), mainly for Bengali and French, but they propose it also for Hindi and Gujarati. The method finds clusters of similar words in a corpus. The clusters are called stems. The method works best for languages that are basically suffix based. For Bengali precision was 39.3 percent better than without stemming, though no absolute numbers were reported for precision. The system was trained on a corpus containing 301 562 words.

Kanis & Müller (2005) used an automatic technique called OOV Words Lemmatization to train their lemmatizer on Czech, Finnish and English data. Their algorithm uses two pattern tables to handle suffixes as well as prefixes. Plisson *et al*. (2004) presented results for a system using Ripple Down Rules (RDR) to generate lemmatization rules for Slovene, achieving up to 77 percent accuracy. Matjaž *et al*. (2007) present an RDR system producing efficient suffix based lemmatizers for 14 languages, three of which (English, German and Slovene) our algorithm also has been tested with.

Stempel (Białecki 2004) is a stemmer for Polish that is trained on Polish full form – lemma pairs. When tested with inflected out-of-vocabulary (OOV) words Stempel produces 95.4 percent correct stems, of which about 81 percent also happen to be correct lemmas.

Hedlund (2001) used two different approaches to automatically find stemming rules from a corpus, for both Swedish and English. Unfortunately neither of these approaches did beat the hand crafted rules in the Porter stemmer for English (Porter 1980) or the Euroling SiteSeeker stemmer for Swedish, (Carlberger *et al*. 2001).

Jongejan & Haltrup (2005) constructed a trainable lemmatizer for the lexicographical task of finding lemmas outside the existing dictionary, bootstrapping from a training set of full form – lemma pairs extracted from the existing dictionary. This lemmatizer looks only at the suffix part of the word. Its performance was compared with a stemmer using hand crafted stemming rules, the Euroling SiteSeeker stemmer for Swedish, Danish and Norwegian, and also with a stemmer for Greek, (Dalianis & Jongejan 2006). The results showed that lemmatizer was as good as the stemmer for Swedish, slightly better for Danish and Norwegian but worse for Greek. These results are very dependent on the quality (errors, size) and complexity (diacritics, capitals) of the training data.

In the current work we have used Jongejan & Haltrup's lemmatizer as a reference, referring to it as the 'suffix lemmatizer'.

3 Delineation

3.1 Why affix rules?

German and Dutch need more advanced methods than suffix replacement since their affixing of words (inflection of words) can include both prefixing, infixing and suffixing. Therefore we created a trainable lemmatizer that handles pre- and infixes in addition to suffixes.

Here is an example to get a quick idea of what we wanted to achieve with the new training algorithm. Suppose we have the following Dutch full form – lemma pair:

`afgevraagd → afvragen`

(Translation: `wondered, to wonder`)

If this were the sole input given to the training program, it should produce a transformation rule like this:

`*ge*a*d → ***en`

The asterisks are wildcards and placeholders. The pattern on the left hand side contains three wildcards, each one corresponding to one placeholder in the replacement string on the right hand side, in the same order. The characters matched by a wildcard are inserted in the place kept free by the corresponding placeholder in the replacement expression.

With this "set" of rules a lemmatizer would be able to construct the correct lemma for some words that had not been used during the training, such as the word `verstekgezaagd` (Translation: `mitre cut`):

Word	**verstek**	**ge**	**z**	**a**	**ag**	**d**
Pattern	*	ge	*	a	*	d
Replacement	*		*		*	en
Lemma	**verstek**		**z**		**ag**	**en**

Table 1. Application of a rule to an OOV word.

For most words, however, the lemmatizer would simply fail to produce any output, because not all words do contain the literal strings `ge` and `a` and a final `d`. We remedy this by adding a one-size-fits-all rule that says "return the input as output":

`* → *`

So now our rule set consists of two rules:

```
*ge*a*d → ***en
* → *
```

The lemmatizer then finds the rule with the most specific pattern (see 4.2) that matches and applies only this rule. The last rule's pattern matches any word and so the lemmatizer cannot fail to produce output. Thus, in our toy rule set consisting of two rules, the first rule handles words like `gevraagd`, `afgezaagd`, `geklaagd`, (all three correctly) and `getalmd` (incorrectly) while the second rule handles words like `directeur` (correctly) and `zei` (incorrectly).

3.2 Inflected *vs.* agglutinated languages

A lemmatizer that only applies one rule per word is useful for inflected languages, a class of languages that includes all Indo-European languages. For these languages morphological change is not a productive process, which means that no word can be morphologically changed in an unlimited number of ways. Ideally, there are only a finite number of inflection schemes and thus a finite number of lemmatization rules should suffice to lemmatize indefinitely many words.

In agglutinated languages, on the other hand, there are classes of words that in principle have innumerous word forms. One way to lemmatize such words is to peel off all agglutinated morphemes one by one. This is an iterative process and therefore the lemmatizer discussed in this paper, which applies only one rule per word, is not an obvious choice for agglutinated languages.

3.3 Supervised training

An automatic process to create lemmatization rules is described in the following sections. By reserving a small part of the available training data for testing it is possible to quite accurately estimate the probability that the lemmatizer would produce the right lemma given any unknown word belonging to the language, even without requiring that the user masters the language (Kohavi 1995).

On the downside, letting a program construct lemmatization rules requires an extended list of full form – lemma pairs that the program can exercise on – at least tens of thousands and possibly over a million entries (Dalianis and Jongejan 2006).

3.4 Criteria for success

The main challenge for the training algorithm is that it must produce rules that accurately lemmatize OOV words. This requirement translates to two opposing tendencies during training. On the one hand we must trust rules with a wide basis of training examples more than rules with a small basis, which favours rules with patterns that fit many words. On the other hand we have the incompatible preference for cautious rules with rather specific patterns, because these must be better at avoiding erroneous rule applications than rules with generous patterns. The envisaged expressiveness of the lemmatization rules – allowing all kinds of affixes and an unlimited number of wildcards – turns the challenge into a difficult balancing act.

In the current work we wanted to get an idea of the advantages of an affix-based algorithm compared to a suffix-only based algorithm. Therefore we have made the task as hard as possible by not allowing language specific adaptations to the algorithms and by not subdividing the training words in word classes.

4 Generation of rules and look-up data structure

4.1 Building a rule set from training pairs

The training algorithm generates a data structure consisting of rules that a lemmatizer must traverse to arrive at a rule that is elected to fire.

Conceptually the training process is as follows. As the data structure is being built, the full form in each training pair is tentatively lemmatized using the data structure that has been created up to that stage. If the elected rule produces the right lemma from the full form, nothing needs to be done. Otherwise, the data structure must be expanded with a rule such that the new rule *a)* is elected instead of the erroneous rule and *b)* produces the right lemma from the full form. The training process terminates when the full forms in all pairs in the training set are transformed to their corresponding lemmas.

After training, the data structure of rules is made permanent and can be consulted by a lemmatizer. The lemmatizer must elect and fire rules in the same way as the training algorithm, so that all words from the training set are lemmatized correctly. It may however fail to produce the correct lemmas for words that were not in the training set – the OOV words.

4.2 Internal structure of rules: prime and derived rules

During training the Ratcliff/Obershelp algorithm (Ratcliff & Metzener 1988) is used to find the longest non-overlapping similar parts in a given full form – lemma pair. For example, in the pair

`afgevraagd → afvragen`

the longest common substring is `vra`, followed by `af` and `g`. These similar parts are replaced with wildcards and placeholders:

`*ge*a*d → ***en`

Now we have the prime rule for the training pair, the least specific rule necessary to lemmatize the word correctly. Rules with more specific patterns – derived rules – can be created by adding characters and by removing or adding wildcards. A rule that is derived from another rule (derived or prime) is more specific than the original rule: Any word that is successfully matched by the pattern of a derived rule is also successfully matched by the pattern of the original rule, but the converse is not the case. This establishes a partial ordering of all rules. See Figures 1 and 2, where the rules marked 'p' are prime rules and those marked 'd' are derived.

Innumerous rules can be derived from a rule with at least one wildcard in its pattern, but only a limited number can be tested in a finite time. To keep the number of candidate rules within practical limits, we used the strategy that the pattern of a candidate is minimally different from its parent's pattern: it can have one extra literal character or one wildcard less or replace one wildcard with one literal character. Alternatively, a candidate rule (such as the bottom rule in Figure 4) can arise by merging two rules. Within these constraints, the algorithm creates all possible candidate rules that transform one or more training words to their corresponding lemmas.

4.3 External structure of rules: partial ordering in a DAG and in a tree

We tried two different data structures to store new lemmatizer rules, a directed acyclic graph (DAG) and a plain tree structure with depth first, left to right traversal.

The DAG (Figure 1) expresses the complete partial ordering of the rules. There is no preferential order between the children of a rule and all paths away from the root must be regarded as equally valid. Therefore the DAG may lead to several lemmas for the same input word. For example, without the rule in the bottom part of Figure 1, the word `gelopen` would have been lemmatized to both `lopen` (correct) and `gelopen` (incorrect):

```
gelopen:
*ge* → **          lopen
*pen → *pen        gelopen
```

By adding a derived rule as a descendent of both these two rules, we make sure that lemmatization of the word `gelopen` is only handled by one rule and only results in the correct lemma:

```
gelopen:
*ge*pen → **pen    lopen
```

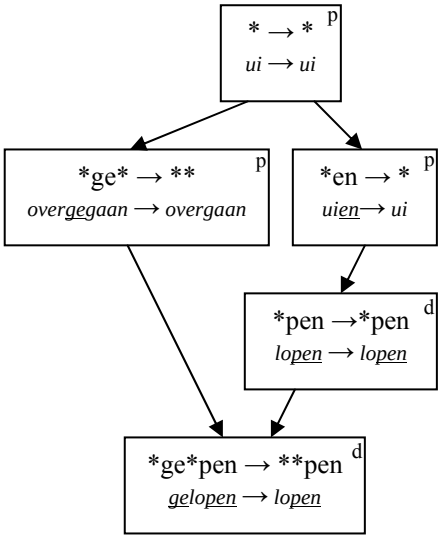

Figure 1. Five training pairs as supporters for five rules in a DAG.

The tree in Figure 2 is a simpler data structure and introduces a left to right preferential order between the children of a rule. Only one rule fires and only one lemma per word is produced. For example, because the rule `*ge* → **` precedes its sibling rule `*en → *`, whenever the former rule is applicable, the latter rule and its descendents are not even visited, irrespective of their applicability. In our example, the former rule – and only the former rule – handles the lemmatization of `gelopen`, and since it produces the correct lemma an additional rule is not necessary.

In contrast to the DAG, the tree implements negation: if the N^{th} sibling of a row of children fires, it not only means that the pattern of the N^{th} rule matches the word, it also means that the patterns of the N-1 preceding siblings do not match the word. Such implicit negation is not possible in the DAG, and this is probably the main reason why the experiments with the DAG-structure lead to huge numbers of rules, very little gener-

alization, uncontrollable training times (months, not minutes!) and very low lemmatization quality. On the other hand, the experiments with the tree structure were very successful. The building time of the rules is acceptable, taking small recursive steps during the training part. The memory use is tractable and the quality of the results is good provided good training material.

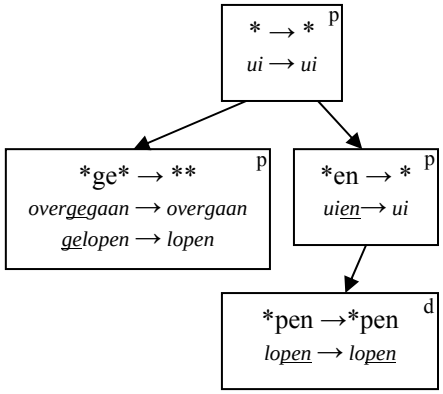

Figure 2. The same five training pairs as supporters for only four rules in a tree.

4.4 Rule selection criteria

This section pertains to the training algorithm employing a tree.

The typical situation during training is that a rule that already has been added to the tree makes lemmatization errors on some of the training words. In that case one or more corrective children have to be added to the rule[1].

If the pattern of a new child rule only matches some, but not all training words that are lemmatized incorrectly by the parent, a right sibling rule must be added. This is repeated until all training words that the parent does not lemmatize correctly are matched by the leftmost child rule or one of its siblings.

A candidate child rule is faced with training words that the parent did not lemmatize correctly and, surprisingly, also supporters of the parent, because the pattern of the candidate cannot discriminate between these two groups.

On the output side of the candidate appear the training pairs that are lemmatized correctly by the candidate, those that are lemmatized incorrectly and those that do not match the pattern of the candidate.

For each candidate rule the training algorithm creates a 2×3 table (see Table 2) that counts the number of training pairs that the candidate lemmatizes correctly or incorrectly or that the candidate does not match. The two columns count the training pairs that, respectively, were lemmatized incorrectly and correctly by the parent. These six parameters N_{xy} can be used to select the best candidate. Only four parameters are independent, because the numbers of training words that the parent lemmatized incorrectly (N_w) and correctly (N_r) are the same for all candidates. Thus, after the application of the first and most significant selection criterion, up to three more selection criteria of decreasing significance can be applied if the preceding selection ends in a tie.

Parent / Child	Incorrect	Correct (supporters)
Correct	$\boldsymbol{N_{wr}}$	N_{rr}
Incorrect	N_{ww}	$\boldsymbol{N_{rw}}$
Not matched	N_{wn}	N_{rn}
Sum	N_w	N_r

Table 2. The six parameters for rule selection among candidate rules.

A large N_{wr} and a small N_{rw} are desirable. N_{wr} is a measure for the rate at which the updated data structure has learned to correctly lemmatize those words that previously were lemmatized incorrectly. A small N_{rw} indicates that only few words that previously were lemmatized correctly are spoiled by the addition of the new rule. It is less obvious how the other numbers weigh in.

We have obtained the most success with criteria that first select for highest $N_{wr} + N_{rr} - N_{rw}$. If the competition ends in a tie, we select for lowest N_{rr} among the remaining candidates. If the competition again ends in a tie, we select for highest $N_{rn} - N_{ww}$. Due to the marginal effect of a fourth criterion we let the algorithm randomly select one of the remaining candidates instead.

The training pairs that are matched by the pattern of the winning rule become the supporters and non-supporters of that new rule and are no longer supporters or non-supporters of the parent. If the parent still has at least one non-supporter, the remaining supporters and non-supporters – the training pairs that the winning

[1] If the case of a DAG, care must be taken that the complete representation of the partial ordering of rules is maintained. Any new rule not only becomes a child of the rule that it was aimed at as a corrective child, but often also of several other rules.

candidate does not match – are used to select the right sibling of the new rule.

5 Evaluation

We trained the new lemmatizer using training material for Danish (STO), Dutch (CELEX), English (CELEX), German (CELEX), Greek (Petasis *et al.* 2003), Icelandic (IFD), Norwegian (SCARRIE), Polish (Morfologik), Slovene (Juršič *et al*. 2007) and Swedish (SUC).

The guidelines for the construction of the training material are not always known to us. In some cases, we know that the full forms have been generated automatically from the lemmas. On the other hand, we know that the Icelandic data is derived from a corpus and only contains word forms occurring in that corpus. Because of the uncertainties, the results cannot be used for a quantitative comparison of the accuracy of lemmatization between languages.

Some of the resources were already disambiguated (one lemma per full form) when we received the data. We decided to disambiguate the remaining resources as well. Handling homographs wisely is important in many lemmatization tasks, but there are many pitfalls. As we only wanted to investigate the improvement of the affix algorithm over the suffix algorithm, we decided to factor out ambiguity. We simply chose the lemma that comes first alphabetically and discarded the other lemmas from the available data.

The evaluation was carried out by dividing the available material in training data and test data in seven different ratios, setting aside between 1.54% and 98.56% as training data and the remainder as OOV test data. (See section 7). To keep the sample standard deviation *s* for the accuracy below an acceptable level we used the evaluation method *repeated random subsampling validation* that is proposed in Voorhees (2000) and Bouckaert & Frank (2000). We repeated the training and evaluation for each ratio with several randomly chosen sets, up to 17 times for the smallest and largest ratios, because these ratios lead to relatively small training sets and test sets respectively. The same procedure was followed for the suffix lemmatizer, using the same training and test sets. Table 3 shows the results for the largest training sets.

For some languages lemmatization accuracy for OOV words improved by deleting rules that are based on very few examples from the training data. This pruning was done after the training of the rule set was completed. Regarding the affix algorithm, the results for half of the languages became better with mild pruning, i.e. deleting rules with only one example. For Danish, Dutch, German, Greek and Icelandic pruning did not improve accuracy. Regarding the suffix algorithm, only English and Swedish profited from pruning.

Language	Suffix %	Affix %	Δ %	$N \times 1000$	*n*
Icelandic	73.2±1.4	71.3±1.5	-1.9	58	17
Danish	93.2±0.4	92.8±0.2	-0.4	553	5
Norwegian	87.8±0.4	87.6±0.3	-0.2	479	6
Greek	90.2±0.3	90.4±0.4	0.2	549	5
Slovene	86.0±0.6	86.7±0.3	0.7	199	9
Swedish	91.24±0.18	92.3±0.3	1.0	478	6
German	90.3±0.5	91.46±0.17	1.2	315	7
English	87.5±0.9	89.0±1.3	1.5	76	15
Dutch	88.2±0.5	90.4±0.5	2.3	302	7
Polish	69.69±0.06	93.88±0.08	24.2	3443	2

Table 3. Accuracy for the suffix and affix algorithms. The fifth column shows the size of the available data. Of these, 98.56% was used for training and 1.44% for testing. The last column shows the number *n* of performed iterations, which was inversely proportional to $\sqrt{N}$ with a minimum of two.

6 Some language specific notes

For Polish, the suffix algorithm suffers from overtraining. The accuracy tops at about 100 000 rules, which is reached when the training set comprises about 1 000 000 pairs.

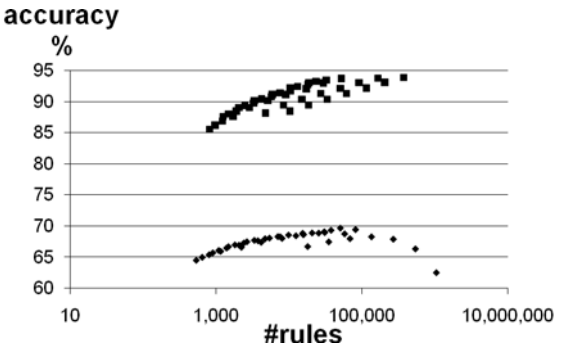

Figure 3. Accuracy vs. number of rules for Polish
Upper swarm of data points: affix algorithm.
Lower swarm of data points: suffix algorithm.
Each swarm combines results from six rule sets with varying amounts of pruning (no pruning and pruning with cut-off = 1..5).

If more training pairs are added, the number of rules grows, but the accuracy falls. The affix algorithm shows no sign of overtraining, even

though the Polish material comprised 3.4 million training pairs, more than six times the number of the second language on the list, Danish. See Figure 3.

The improvement of the accuracy for Polish was tremendous. The inflectional paradigm in Polish (as in other Slavic languages) can be left factorized, except for the superlative. However, only 3.8% of the words in the used Polish data have the superlative forming prefix `naj`, and moreover this prefix is only removed from adverbs and not from the much more numerous adjectives.

The true culprit of the discrepancy is the great number (> 23%) of words in the Polish data that have the negative prefix `nie`, which very often does not recur in the lemma. The suffix algorithm cannot handle these 23% correctly.

The improvement over the suffix lemmatizer for the case of German is unassuming. To find out why, we looked at how often rules with infix or prefix patterns fire and how well they are doing. We trained the suffix algorithm with 9/10 of the available data and tested with the remaining 1/10, about 30 000 words. Of these, 88% were lemmatized correctly (a number that indicates the smaller training set than in Table 3).

	German		Dutch	
	Acc. %	Freq %	Acc. %	Freq %
all	88.1	100.0	87.7	100.0
suffix-only	88.7	94.0	88.1	94.9
prefix	79.9	4.4	80.9	2.4
infix	83.3	2.3	77.4	3.0
ä ö ü	92.8	0.26	N/A	0.0
ge infix	68.6	0.94	77.9	2.6

Table 4. Prevalence of suffix-only rules, rules specifying a prefix, rules specifying an infix and rules specifying infixes containing either *ä*, *ö* or *ü* or the letter combination *ge*.

Almost 94% of the lemmas were created using suffix-only rules, with an accuracy of almost 89%. Less than 3% of the lemmas were created using rules that included at least one infix subpattern. Of these, about 83% were correctly lemmatized, pulling the average down. We also looked at two particular groups of infix-rules: those including the letters `ä`, `ö` or `ü` and those with the letter combination `ge`. The former group applies to many words that display umlaut, while the latter applies to past participles. The first group of rules, accounting for 11% of all words handled by infix rules, performed better than average, about 93%, while the latter group, accounting for 40% of all words handled by infix rules, performed poorly at 69% correct lemmas. Table 4 summarizes the results for German and the closely related Dutch language.

7 Self-organized criticality

Over the whole range of training set sizes the number of rules goes like $C.N^d$ with $0 < C$, and N the number of training pairs. The value of C and d not only depended on the chosen algorithm, but also on the language. Figure 4 shows how the number of generated lemmatization rules for Polish grows as a function of the number of training pairs.

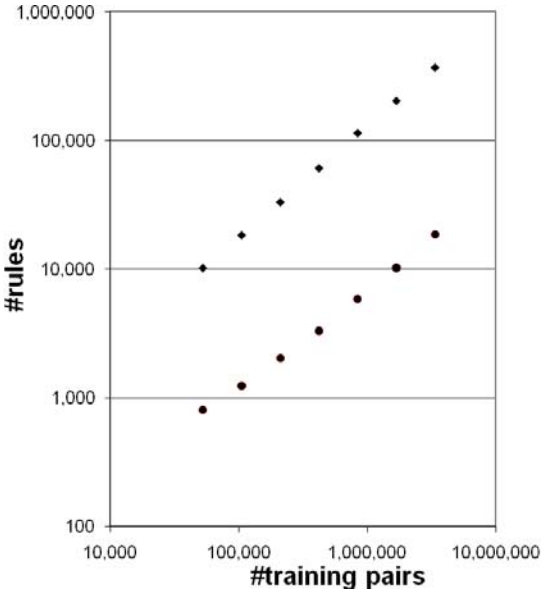

Figure 4. Number of rules vs. number of training pairs for Polish (double logarithmic scale).
Upper row: unpruned rule sets
Lower row: heavily pruned rule sets (cut-off=5)

There are two rows of data, each row containing seven data points. The rules are counted after training with 1.54 percent of the available data and then repeatedly doubling to 3.08, 6.16, 12.32, 24.64, 49.28 and 98.56 percent of the available data. The data points in the upper row designate the number of rules resulting from the training process. The data points in the lower row arise by pruning rules that are based on less than six examples from the training set.

The power law for the upper row of data points for Polish in Figure 4 is

$$N_{rules} = 0.80 N_{training}^{0.87}$$

As a comparison, for Icelandic the power law for the unpruned set of rules is

$$N_{rules} = 1.32 N_{training}^{0.90}$$

These power law expressions are derived for the affix algorithm. For the suffix algorithm the exponent in the Polish power law expression is very close to 1 (0.98), which indicates that the suffix lemmatizer is not good at all at generalizing over the Polish training data: the number of rules grows almost proportionally with the number of training words. (And, as Figure 3 shows, to no avail.) On the other hand, the suffix lemmatizer fares better than the affix algorithm for Icelandic data, because in that case the exponent in the power law expression is lower: 0.88 versus 0.90.

The power law is explained by self-organized criticality (Bak *et al.* 1987, 1988). Rule sets that originate from training sets that only differ in a single training example can be dissimilar to any degree depending on whether and where the difference is tipping the balance between competing rule candidates. Whether one or the other rule candidate wins has a very significant effect on the parts of the tree that emanate as children or as siblings from the winning node. If the difference has an effect close to the root of the tree, a large expanse of the tree is affected. If the difference plays a role closer to a leaf node, only a small patch of the tree is affected. The effect of adding a single training example can be compared with dropping a single rice corn on top of a pile of rice, which can create an avalanche of unpredictable size.

8 Conclusions

Affix rules perform better than suffix rules if the language has a heavy pre- and infix morphology and the size of the training data is big. The new algorithm worked very well with the Polish Morfologik dataset and compares well with the Stempel algorithm (Białecki 2008).

Regarding Dutch and German we have observed that the affix algorithm most often applies suffix-only rules to OOV words. We have also observed that words lemmatized this way are lemmatized better than average. The remaining words often need morphological changes in more than one position, for example both in an infix and a suffix. Although these changes are correlated by the inflectional rules of the language, the number of combinations is still large, while at the same time the number of training examples exhibiting such combinations is relatively small. Therefore the more complex rules involving infix or prefix subpatterns or combinations thereof are less well-founded than the simple suffix-only rules. The lemmatization accuracy of the complex rules will therefore in general be lower than that of the suffix-only rules. The reason why the affix algorithm is still better than the algorithm that only considers suffix rules is that the affix algorithm only generates suffix-only rules from words with suffix-only morphology. The suffix-only algorithm is not able to generalize over training examples that do not fulfil this condition and generates many rules based on very few examples. Consequently, everything else being equal, the set of suffix-only rules generated by the affix algorithm must be of higher quality than the set of rules generated by the suffix algorithm.

The new affix algorithm has fewer rules supported by only one example from the training data than the suffix algorithm. This means that the new algorithm is good at generalizing over small groups of words with exceptional morphology. On the other hand, the bulk of 'normal' training words must be bigger for the new affix based lemmatizer than for the suffix lemmatizer. This is because the new algorithm generates immense numbers of candidate rules with only marginal differences in accuracy, requiring many examples to find the best candidate.

When we began experimenting with lemmatization rules with unrestricted numbers of affixes, we could not know whether the limited amount of available training data would be sufficient to fix the enormous amount of free variables with enough certainty to obtain higher quality results than obtainable with automatically trained lemmatizers allowing only suffix transformations.

However, the results that we have obtained with the new affix algorithm are on a par with or better than those of the suffix lemmatizer. There is still room for improvements as only part of the parameter space of the new algorithm has been searched. The case of Polish shows the superiority of the new algorithm, whereas the poor results for Icelandic, a suffix inflecting language with many inflection types, were foreseeable, because we only had a small training set.

9 Future work

Work with the new affix lemmatizer has until now focused on the algorithm. To really know if the carried out theoretical work is valuable we would like to try it out in a real search setting in a search engine and see if the users appreciate the new algorithm's results.

References

Per Bak, Chao Tang and Kurt Wiesenfeld. 1987. Self-Organized Criticality: An Explanation of 1/f Noise, *Phys. Rev. Lett.*, vol. 59,. pp. 381-384, 1987

Per Bak, Chao Tang and Kurt Wiesenfeld . 1988. *Phys. Rev.* A38, (1988), pp. 364-374

Andrzej Białecki, 2004, Stempel - Algorithmic Stemmer for Polish Language http://www.getopt.org/stempel/

Remco R. Bouckaert and Eibe Frank. 2000. Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. In H. Dai, R. Srikant, & C. Zhang (Eds.), Proc. 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia, May 26-28, 2004 (pp. 3-12). Berlin: Springer.

Johan Carlberger, Hercules Dalianis, Martin Hassel, and Ola Knutsson. 2001. Improving Precision in Information Retrieval for Swedish using Stemming. In the *Proceedings of NoDaLiDa-01 - 13th Nordic Conference on Computational Linguistics*, May 21-22, Uppsala, Sweden.

Celex: http://celex.mpi.nl/

Hercules Dalianis and Bart Jongejan 2006. Handcrafted versus Machine-learned Inflectional Rules: the Euroling-SiteSeeker Stemmer and CST's Lemmatiser, in *Proceedings of the International Conference on Language Resources and Evaluation*, LREC 2006.

F. Çuna Ekmekçioglu, Mikael F. Lynch, and Peter Willett. 1996. Stemming and N-gram matching for term conflation in Turkish texts. *Information Research*, 7(1) pp 2-6.

Niklas Hedlund 2001. *Automatic construction of stemming rules*, Master Thesis, NADA-KTH, Stockholm, TRITA-NA-E0194.

IFD: Icelandic Centre for Language Technology, http://tungutaekni.is/researchsystems/rannsoknir_12en.html

Bart Jongejan and Dorte Haltrup. 2005. *The CST Lemmatiser*. Center for Sprogteknologi, University of Copenhagen version 2.7 (August, 23 2005) http://cst.dk/online/lemmatiser/cstlemma.pdf

Jakub Kanis and Ludek Müller. 2005. Automatic Lemmatizer Construction with Focus on OOV Words Lemmatization in Text, Speech and Dialogue, *Lecture Notes in Computer Science*, Berlin / Heidelberg, pp 132-139

Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* 2 (12): 1137–1143, Morgan Kaufmann, San Mateo.

Prasenjit Majumder, Mandar Mitra, Swapan K. Parui, Gobinda Kole, Pabitra Mitra, and Kalyankumar Datta. 2007. YASS: Yet another suffix stripper. *ACM Transactions on Information Systems* , Volume 25 , Issue 4, October 2007.

Juršič Matjaž, Igor Mozetič, and Nada Lavrač. 2007. Learning ripple down rules for efficient lemmatization In *proceeding of the Conference on Data Mining and Data Warehouses* (SiKDD 2007), October 12, 2007, Ljubljana, Slovenia

Morfologik: Polish morphological analyzer http://mac.softpedia.com/get/Word-Processing/Morfologik.shtml

Douglas W. Oard, Gina-Anne Levow, and Clara I. Cabezas. 2001. CLEF experiments at Maryland: Statistical stemming and backoff translation. In Cross-language information retrieval and evaluation: *Proceeding of the Clef 2000 workshops* Carol Peters Ed. Springer Verlag pp. 176-187. 2001.

Georgios Petasis, Vangelis Karkaletsis , Dimitra Farmakiotou , Ion Androutsopoulos and Constantine D. Spyropoulo. 2003. A Greek Morphological Lexicon and its Exploitation by Natural Language Processing Applications. In Lecture Notes on Computer Science (LNCS), vol.2563, "Advances in Informatics - Post-proceedings of the 8th Panhellenic Conference in Informatics", Springer Verlag.

Joël Plisson, Nada Lavrač, and Dunja Mladenic. 2004, A rule based approach to word lemmatization, *Proceedings of the 7th International Multiconference Information Society*, IS-2004, Institut Jozef Stefan, Ljubljana, pp.83-6.

Martin F. Porter 1980. An algorithm for suffix stripping. *Program*, vol 14, no 3, pp 130-130.

John W. Ratcliff and David Metzener, 1988. Pattern Matching: The Gestalt Approach, *Dr. Dobb's Journal*, page 46, July 1988.

SCARRIE 2009. Scandinavian Proofreading Tools http://ling.uib.no/~desmedt/scarrie/

STO: http://cst.ku.dk/sto_ordbase/

SUC 2009. Stockholm Umeå corpus, http://www.ling.su.se/staff/sofia/suc/suc.html

Pieter Theron and Ian Cloete 1997 Automatic acquisition of two-level morphological rules, *Proceedings of the fifth conference on Applied natural language processing*, p.103-110, March 31-April 03, 1997, Washington, DC.

Ellen M. Voorhees. 2000. Variations in relevance judgments and the measurement of retrieval effectiveness, J. of Information Processing and Management 36 (2000) pp 697-716